BYOP PROPOSAL

Project Domain : Computer Vision

Abhivansh Gupta
23112004
Chemical Engineering
abhivansh_g@ch.iitr.ac.in
+91 8302905563
20 October 2024

Problem Description:

In public and private spaces, **surveillance systems** are crucial for maintaining security. They generate enormous amounts of video data, much of which contains normal, everyday activities. However, **anomalous events**—such as theft, violence, or unauthorized access—may occur infrequently but require immediate attention. The sheer volume of surveillance footage makes it infeasible for human operators to manually monitor everything in real time, leading to potential delays in identifying critical events.

Current surveillance solutions often rely on motion-based alerts or static rule-based systems, which result in **false alarms** or missed detections in dynamic environments where subtle anomalous behaviors go unnoticed. These systems struggle with complex scenarios where anomalies are context-dependent and hard to define, such as recognizing suspicious behavior in a crowd or identifying unusual activity in low-visibility conditions.

AI/ML Involvement:

1. Definition and Purpose

- **Anomaly Detection**: The goal is to identify unusual patterns or events in video data that do not conform to expected behavior, which could indicate security threats, accidents, or other significant incidents.
- **Surveillance Systems**: These systems continuously monitor areas for safety and security, making real-time anomaly detection essential.

2. AI and ML Techniques

- **Computer Vision**: ML algorithms analyze video frames to detect and track objects, enabling the system to understand the scene context and identify deviations.
- **Feature Extraction**: Algorithms extract relevant features (e.g., motion patterns, object behavior) from video data to distinguish normal activities from anomalies.
- **Statistical Analysis**: Statistical models assess the likelihood of observed events, flagging those that deviate significantly from the norm.

Problem Breakdown:

Aim:

To develop a system that detects anomalous activities in video surveillance footage using **Temporal Convolutional Networks (TCNs)**. The project will focus on the temporal structure of videos to identify rare or unusual events (anomalies) in a sequence of frames, leveraging the **UCF-Crime** dataset.

Challenges:

Data Quality: High-quality labeled datasets are crucial for effective model training but can be difficult to obtain.
Real-Time Processing: Surveillance systems require models that can operate in real-time, which poses challenges in computation and resource management.
Variability: Anomalies can vary significantly based on context, lighting, and scene complexity, making detection challenging.

Tools and Technologies:

- **Frameworks**: PyTorch
- **Preprocessing**: OpenCV for video processing, frame extraction, and augmentation.
- **Visualization**: Matplotlib and TensorBoard for tracking performance and visualizing results.
- **Real-time Inference**: OpenCV for live video stream handling, and integrate it with the trained TCN model.

Key Goals:

1. Efficiently preprocess and handle video data.
2. Build a model that can capture temporal dependencies between video frames.
3. Detect anomalous events with high precision and accuracy.
4. Optimize the model for computational efficiency given large-scale video data.

Dataset:

UCF-Crime Dataset:

- Contains videos with labels indicating normal or anomalous activities.
- Publicly available and includes multiple scenarios such as burglary, fighting, shooting, and normal events like walking, standing, etc.
- Dataset link: https://www.kaggle.com/datasets/odins0n/ucf-crime-dataset

---

TIMELINE FOR THE PROJECT

Week 1: Research and Data Preprocessing

1.1 Research and Planning:

- **Study Anomaly Detection Techniques**:
  - Reviewing anomaly detection techniques in video, especially those using **TCNs**, **3D CNNs**, or **Recurrent Neural Networks (RNNs)**.
  - Key papers to review:
    - "Anomaly Detection in Video Surveillance Using Deep Learning" (TCN-based approach).
    - "Deep Learning for Anomaly Detection in Surveillance Videos" (Comparing CNN, RNN, TCN).
- **Defining Model Scope**:
  - Selecting a primary model (TCN) with potential additional variants (3D CNNs).
  - Defining performance metrics like **precision, recall, F1-score**, and **ROC AUC** for evaluation.

1.2 Data Preprocessing:

- **Frame Extraction**:
  - Extracting frames from each video at a fixed frame rate (e.g., 10 fps) to reduce data redundancy.
  - Applying **data augmentation** (random crops, flips, slight rotations) for training generalization.
- **Feature Extraction**:
  - Using **pretrained 2D CNN** (e.g., **ResNet50**, **InceptionV3**) to extract spatial features from frames.
  - Converting each frame into a **feature vector** and save them for use in the temporal model (TCN).

1.3 Setup Data Pipeline:

- **Data Generator**:
  - Building an efficient data pipeline to load sequences of frames and their corresponding labels (normal/anomalous).
  - Converting video frames into **feature sequences** for the temporal model.

Week 2: Model Development (TCN)

2.1 Temporal Convolutional Network (TCN) Architecture:

- **Defining TCN Architecture**:
  - Input: Sequence of **feature vectors** extracted from frames.
  - **Temporal Convolutional Layers**: Apply 1D convolutions over the time axis to capture temporal dependencies between consecutive frames.
  - **Causal Convolutions**: Ensure that future information is not leaked into past events.
  - **Dilated Convolutions**: Use dilations to allow TCNs to handle long-range dependencies in video sequences.
- **Modellling Layers**:
  - **Input**: Sequencing of frame features (each feature is derived from a CNN).
  - **Temporal Convolution Layers**: Stacking multiple TCN layers with increasing dilation factors.
  - **Dropout**: Regularising the model to prevent overfitting.
  - **Fully Connected Layers**: Converting the temporal output into a final binary classification (normal/anomaly).

2.2 Implement and Train Baseline Model:

- **Training**:
  - Training the model
  - Using **sequence batches** from the dataset and label them as normal or anomalous.
- **Evaluation**:
  - Tracking training and validation loss.
  - Using metrics like **accuracy, precision, recall, and F1-score** to evaluate the baseline model.
  - Visualizing the learning process using tools like **TensorBoard** or **Matplotlib**.

Week 3: Advanced Techniques and Model Tuning

3.1 Model Tuning and Hyperparameter Optimization:

- **Hyperparameter Tuning**:
  - Tuning critical parameters like batch size, learning rate, dropout rate, and the number of TCN layers.
  - Experimenting with different dilation factors and kernel sizes in TCN layers.
- **Advanced Data Augmentation**:
  - Adding temporal augmentation techniques like frame shuffling or skipping to enhance model generalization.

3.2 Regularization and Overfitting Control:

- **Dropout**: Regularizing the model to avoid overfitting on the training data.

- **L2 Weight Decay**: Applying weight regularization to control model complexity.

3.3 Incorporating Attention Mechanisms:

- **Attention Layer**:
  - Integrating an **attention mechanism** to help the model focus on important temporal segments (e.g., an anomaly occurring over a few frames).
  - Experimenting with different attention mechanisms (soft vs. hard attention).

3.4 Testing on Unseen Videos:

- **Generalization Testing**:
  - Testing the model on unseen videos from the dataset and assess performance.
  - Analysing **false positives** and **false negatives** to further fine-tune the model.

Week 4: Final Refinement, Evaluation, and Reporting

4.1 Final Model Training and Testing:

- **Full Model Training**:
  - Training the final TCN model on the full UCF-Crime dataset using optimised hyperparameters.
  - Evaluating on a validation set to ensure model robustness.
- **Advanced Metrics**:
  - Measuring the model's **ROC AUC**, precision-recall curve, and confusion matrix to assess performance.
  - Using **frame-level evaluation** to determine when anomalies occur.

4.2 Real-time Implementation:

- **Real-time Adaptation**:
  - Implementing the model for real-time anomaly detection by processing video streams.
  - Creating a simple interface to flag detected anomalies in live video feeds.
- **Model Optimization**:
  - Reducing model complexity for faster inference.
  - Using techniques like **quantization** or **pruning** to speed up detection.

4.3 Final Presentation and Documentation:

- **Visualization**:
  - Showing visualizations of anomaly detection on video clips.

- Plotting **attention heatmaps** to demonstrate which frames the model focused on when detecting anomalies.

If time permits, I will deploy it on Streamlit.

Final Deliverables:

1. **Trained TCN Model**: Capable of detecting anomalies in videos.
2. **Real-time Video Processing Demo**: Demonstration of the model processing live video or video streams.
3. **Evaluation Metrics**: Comprehensive report on model performance (precision, recall, F1-score, etc.).
4. **Project Report**: A complete documentation of the project workflow, including challenges and solutions.
